Development of a videogame using cel-shading in Unity3D
Final Degree Work Report

Kendra Muñoz Arnau

Final Degree Work
Bachelor's Degree in
Video Game Design and Development
Universitat Jaume I

July 1, 2021

Supervised by: Begoña Martinez Salvador

*

To my family, especially Miguel, Thank you for all your support during these tough four years.
Thanks to you I am the best version of myself.

Acknowledgments

First of all, I would like to thank my Final Degree Work supervisor, Begoña Martínez Salvador for all her help and support.

Next, I would like to thank all my friends, especially Miguel Ferrer Carrasco, for the advice given when carrying out this work and the support, and Luna, Cris, Jorge, Carlos and Lara for all the nights together, the talks and the patience of these hard last months.

I also would like to thank Alex, Nuria, Gabri, Irene, Izan, Jorge, Adri, Jordi, Lluc and Pedro for worrying about me, for taking care of me even if I isolate myself for months, for every party night of laughing and crying.

Thanks to Angel, who has always supported me, and has helped me with all my doubts about gamedev from the beginning.

I also want to thank my family for all the things they have done for me, and and Estopa for the music that motivated me in the hardest moments.

And finally, I want to thank José Vte. Martí Avilés for the amazing work done tutoring the subject.

Abstract

This document constitutes the final memory for the Final Degree Project in Videogame Design and Development based on the development of a videogame in Unity 3D, with its own models in cartoon style, using a custom cel-shading. The videogame is a platformer one in which the ultimate goal is to get the fives moons hidden in the level. For this, the player will have to achieve five different goals that unlock the five moons, when the player collects them, the game ends.

Contents

CHAPTER 1

INTRODUCTION

This chapter is an explanation of what was the motivation for the development of this videogame, what I wanted to create and why.

Contents

Mario 64 [6] is one of the most influential 3D platforms videogames since its release. Even today, it is a game that has not become outdated and continues to be played, It has one of the largest communities on game modding and speedruns. On the other hand, The legend of Zelda: Wind Waker [5] has been remembered for its visual style and its own implementation of the Cel shading technique. The combination of this two games is Playroom, the game I have created, designed and implemented in this project. Playroom is a 3rd person 3D platformer for PC in which the player's objective is to overcome all the challenges and collect the 5 moons to beat the game embodying a little girl named Sally, who often has nightmares. In this game, you will live inside one of them.

1.1 Work Motivation

I have always wished to create a 3D platformer video game for many years. Influenced by most famous games like Super Mario 64, one of the main references in platform games, and The Legend of Zelda: Wind Waker, one of the most controversial games for cartoon aesthetics and the use of the cel shading technique of the time, I would like to create a level where it is fun to achieve the proposed objectives of the game.

Programming a shader in Unity is a personal challenge. I have never been involved in shaders before, and although I have studied some aspects in the Computer Graphics Course, I had never dared to work with them in Unity3D. Making my own game, including coding and art is also a challenge for me. My motivation is to understand, even minimally, all the departments in charge of making a video game and how they coalesce between them.

1.2 Related Subjects

- VJ1216 - 3D Design
- VJ1226 - Character Design and Animation
- VJ1227 – Videogame Game Engines
- VJ1222 - Videogame Concept Design
- VJ1223 – Videogame Art
- VJ1203 – Programming I
- VJ1208 – Programming II
- VJ1224 – Software Engineering
- VJ1221– Computer Graphics

1.3 Objectives

- Create a Cel-shader in Unity3D [10].
- Make an 3D videogame, with my own models and programming.
- Modelling an environment in cartoon style.
- Modelling an original character with textures and rig.
- Implementing occlusion culling to optimize the game.

1.4 Environment and Initial State

At first it was going to be an arcade game for mobile, but as the project progressed, I decided to make a 3D platformer combining my two favorite games since it was one of my dreams since my childhood. My initial state was just PC, my graphics tablet, the free Unity3D [10] and Blender [8] programs.

CHAPTER 2

Planning and resources evaluation

Contents

This chapter details the planning and the hardware and software needed while developing the game.

2.1 Planning

Initially, I arranged the planning into 4 groups: modeling, programming, assembling the game in Unity and writing the final report. And I assigned each task more or less the same hours (except for the final memory, which I gave it less hours). Some of the tasks took me more time than planned, such as modeling, because I had to learn everything that 3D in video games entails (see Figure **??**). The final memory also took longer than expected, so the hours changed a little bit (see Figure **??**).

The initial planning is shown in Figure 2.1 and in the Gantt Diagram **??**.

Final planning

I have done a Gantt chart to provide a little summary of how the project went. (see Figure 2.1)

2.2 Resource Evaluation

I work with a fairly old desktop PC. The pieces are 10 years old, although Unity and the game can run without problems. This has caused me the occasional crash of Unity,

Task	**Hours**
Research	
3D in video games industry	6h
Shading in Unity	15h
3D Modeling	
3D mesh	63h
UV's and textures	50h
Programming	
Prototype	70h
Final game	80h
Unity	
Scenes	16h
Documentation	
Technical proposal, game design document and final memory	16h

Table 2.1: Initial planning

Task	**Hours**
Research	
3D in video games industry	12h
Shading in Unity	20h
3D Modeling	
3D mesh	100h
UV's and textures	50h
Programming	
Prototype	50h
Final game	30h
Unity	
Scenes	18h
Documentation	
Technical proposal, game design document and final memory	20h

Table 2.2: Final planning

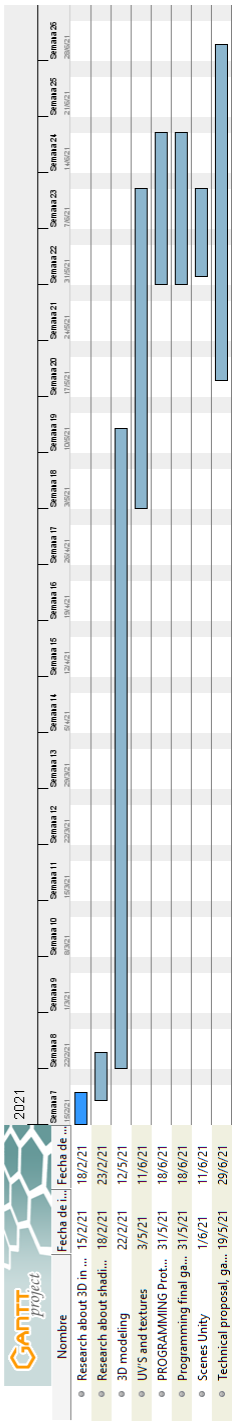

Figure 2.1: Gantt chart (made with GanttProject)

but in the end the game has been optimized. The game has been developed in my PC with the following specifications:

- OS: Windows 10 Pro
- Processor: AMD FX 8350 eight-core
- RAM: 8.0 GB DDR3
- GPU: Nvidia GTX 750 TI

The software that I have used to develop the whole game is:

- Unity: is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as a Mac OS X-exclusive game engine
- Blender: is a free and open-source 3D computer graphics software toolset. I have used Blender for modeling and for texture some smalls assets.
- Substance Painter [3]: 3D painting software that allows you to texture and render 3D meshes. I have used substance painter for texture almost all my assets using a custom smart material.
- Adobe Photoshop [2]: is a raster graphics editor developed and published by Adobe Inc. I have used Photoshop for fixing some errors or adding details to some textures of the game.
- Google Drive: A file storage and synchronization developed by Google.

CHAPTER 3

System Analysis and Design

This chapter details the system analysis and design of the videogame I created, the hardware needed while running the game.

Contents

3.0.1 Functional Requirements

A functional requirement defines a function of the system that is going to be developed. This function is described as a set of inputs, its behavior, and its outputs [11]. The functional requirements can be: calculations, technical details, data manipulations and processes, and any other specific functionality that defines what a system is supposed to achieve.

1. The player will be able to start a new game.
2. The player will be able to move smoothly through the map.
3. The player will be able to quit the game.
4. The player will be able to see the credits.

5. The player will be able to jump.
6. The player will be able to collect moons and coins.
7. The player will be able to press buttons.
8. The player will be able to transition from walking to running.
9. The camera will follow the player smoothly and will avoid going through walls.
10. The game will have five challenges.
11. The game will only have one level.
12. The player will be able to move the camera.
13. The player will be able to run.

The proposed system should allow- The functional requirements related to the query and purchase of credits are shown in Tables from Table 3.1 to Table **??**.

Actor:	Player
Preconditions:	The player must be on the main menu screen.
Steps:	1. The player chooses the option `Start New game` 2. The main scene is load in the window

Table 3.1: Functional requirement «1. Start a new game»

tab:2 2. Move smoothly Player The player must have started a new game

1. The player use the joystick or WASD keys.
2. The character moves in the given direction.

tab:3 3. Exit game. Player The player must have started a new game

1. The player clicks the exit game button.
2. The game ends.

tab:4 4. See the credits Player The player must be on the main menu screen.

1. The player clicks the "about me" button.

2. The game scene changes to the credits scene.

tab:5 5. Jump Player The player must have started a new game

1. The player push A button or space bar key.
2. The character jumps.

tab:6 6. Collect moons Player The player must have started a new game

1. The player stands near the moon.
2. The player approaches the moon, the moon appears in the interface and disappears from the scene

tab:7 7. Collect coins Player The player must have started a new game

1. The player stands near the coin.
2. The player approaches the coin, the coin appears in the interface and disappears from the scene. If the coin number is 5, a moon appears.

Actor:	Player
Preconditions:	1. The player must be in the level. 2. The player must be next to the buttons of the puzzle.
Steps:	1. The player must execute 3 times the use case in Table **??**. 2. The game checks if the sequences of pressed buttons is correct. 3. If it is correct, the player obtains a new moon 4. If not, the puzzle restarts.

Table 3.2: Functional requirement «8. Solve puzzle»

tab:9 9. Run Player The player must have started a new game

1. The player pushes the b button or shift key.
2. The player velocity rise up.

tab:10 10.Move camera Player The player must have started a new game

1. The player moves right joystick or the mouse.
2. The camera moves.

Actor:	Player
Preconditions:	1. The player must be in the level. 2. The player must be next to the buttons of the puzzle.
Steps:	1. The player must jump on the button 2. The button gets pressed and changes its colour 3. The counter of buttons increases by one

Table 3.3: Functional requirement «8. Press buttons»

3.0.2 Non-functional Requirements

Non-functional requirements impose conditions on the design or implementation.

- Efficiency of the interaction and visualization of the videogame so it can run at 24 fps.
- Deliver a visually appealing cartoony look.
- The game will run on a PC.
- The aesthetic will remind of a 2D game.
- The HUD will be simple.
- The system will attend all the requests of the player.

3.1 System Architecture

I don't have any special architecture, the software can work with these requirements:

- OS: Windows 10 Pro
- Processor: AMD FX 8350 eight-core
- RAM: 8.0 GB DDR3
- GPU: Nvidia GTX 750 TI

3.2 Interface Design

The game menus and interfaces are based on The Legend of Zelda: Wind Waker.(see Figure 3.12), (see Figure 3.12), (see Figure 3.12)

Figure 3.1: Moon

The main and pause menus of the game are based on The Legend Of Zelda: Wind Waker HD, colorful and remarkable squares. The pause menu is not a new scene, but a canvas that appears when the Enter key is pressed and completely pauses the game. (see Figure 3.3) The main menu is a new scene (see Figure 3.2)

Figure 3.2: Main Menu

Figure 3.3: Pause Menu

Moons(see Figure 3.1) will be displayed in the top right corner as a row of maximum 5 elements. When you get one, it is added to the row of collected moons displayed on the screen, coins work the same way and they appear in the bottom left of the screen as a row of maximum five.

For the countdown button, once activated, time will appear on the top left. When time is over, it will fade out.

There will also be three menus, which are:

1. Pause menu: During gameplay, pressing the start button will display the pause menu. The game will freeze until the button is pressed again.

2. Title screen: The start menu will consist of three buttons: the “new game” button, the “about me” button, and “quit game” button.

3. Endgame screen: The endgame screen will restart the game.

3.3 Game Flow

The game consists of a single scene that represents a nightmare version of Sally’s bedroom, enlarged and slightly distorted, where 5 moons are hidden and must be collected in order to wake up Sally from her dream. The player can freely explore the room and get the moons, either by finding them or by solving the puzzles that make them appear. There is no scripted order to find the moons, each of them is obtained in a unique way. These are the possible ways of obtaining a moon:

1. Get to the highest platform of the level, which will be the bookshelf.
2. Find the 5 hidden coins that are hidden around the level. They are hidden in these places:
    a) Inside the dresser drawer.
    b) Under the bed.
    c) In the second shelf
    d) In the window frame
    e) On top of the book on the highest shelf
3. Press the colored buttons on the map in the correct order. The solution will be hidden in the level.
4. Time trial. Reach a level point before a time limit. The counter will be activated by pressing a button on the map.
5. Search for the hidden one on the map.

The end of the game is achieved after overcoming the 5 challenges and waking Sally. The following sketch (see Figure 3.4) shows how the moons and red coins are placed in the level.

3.4 Flowchart

A flowchart is a type of diagram that represents the game workflow or process.I have made a representation of how the game works(see Figure 3.5). In addition, I also made a flowchart of how the challenges work(see Figure 3.6).

3.5 Player Control

There will be two different ways to play, keyboard or controller. The keyboard controls will be W A S D keys for movement, spacebar for jumping, shift for running, the mouse will move the camera and intro key will be used to pause the game.(see Figure 3.7)

Using the controller, the movement will be controlled through the left joystick, right joystick for camera movement, A button will be used to jump and B button to run and the start button will pause the game. (see Figure 3.8)

3.6 Visual Style

It is a game with flat colors in a cartoon style. In addition, the Cel-shading technique will be implemented, a shading that is used to simulate 2D cartoon style and its not included by default in Unity, so I have created it from the shaders that Unity offers.

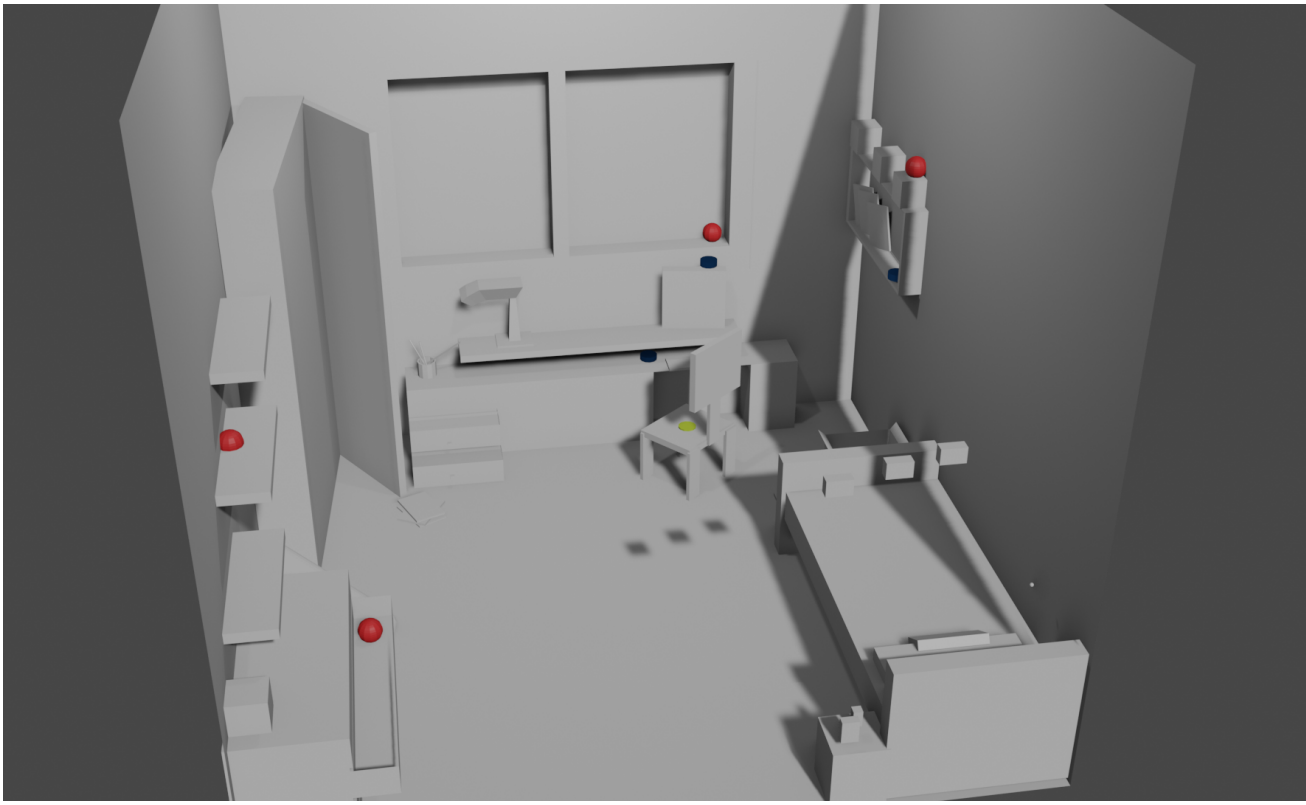

Figure 3.4: 3D sketch of the main scene. The moons are the red pellets of the scene

This type of shading is characterized by flat shading, conventional lighting values are calculated for each pixel and then mapped to a small number of discrete shadows to create the characteristic flat appearance, in which the shadows and the brightest points appear to be blocks of color rather than appear to blend smoothly.

For reference, I'll use The legend of Zelda: Wind Waker(see Figure 3.9) as an example, one of the first games to make the Cel-shading technique and the cartoon style famous in 3D video games.

3.7 Gameplay, story and narrative

As for the gameplay and construction of the game world, the main reference is Mario 64 (see Figure 3.11).

As the world where the game takes place is a nightmare in a little girl's room, the level is a bedroom that is decorated with beautiful objects, but at the same time with some dark details.The context of the game takes place in the head of a little girl named Sally, who has a nightmare in which she finds herself in her own bedroom, shrunken to toy size, something like Alice In Wonderland [4] (see Figure 3.12) when Alice drinks the

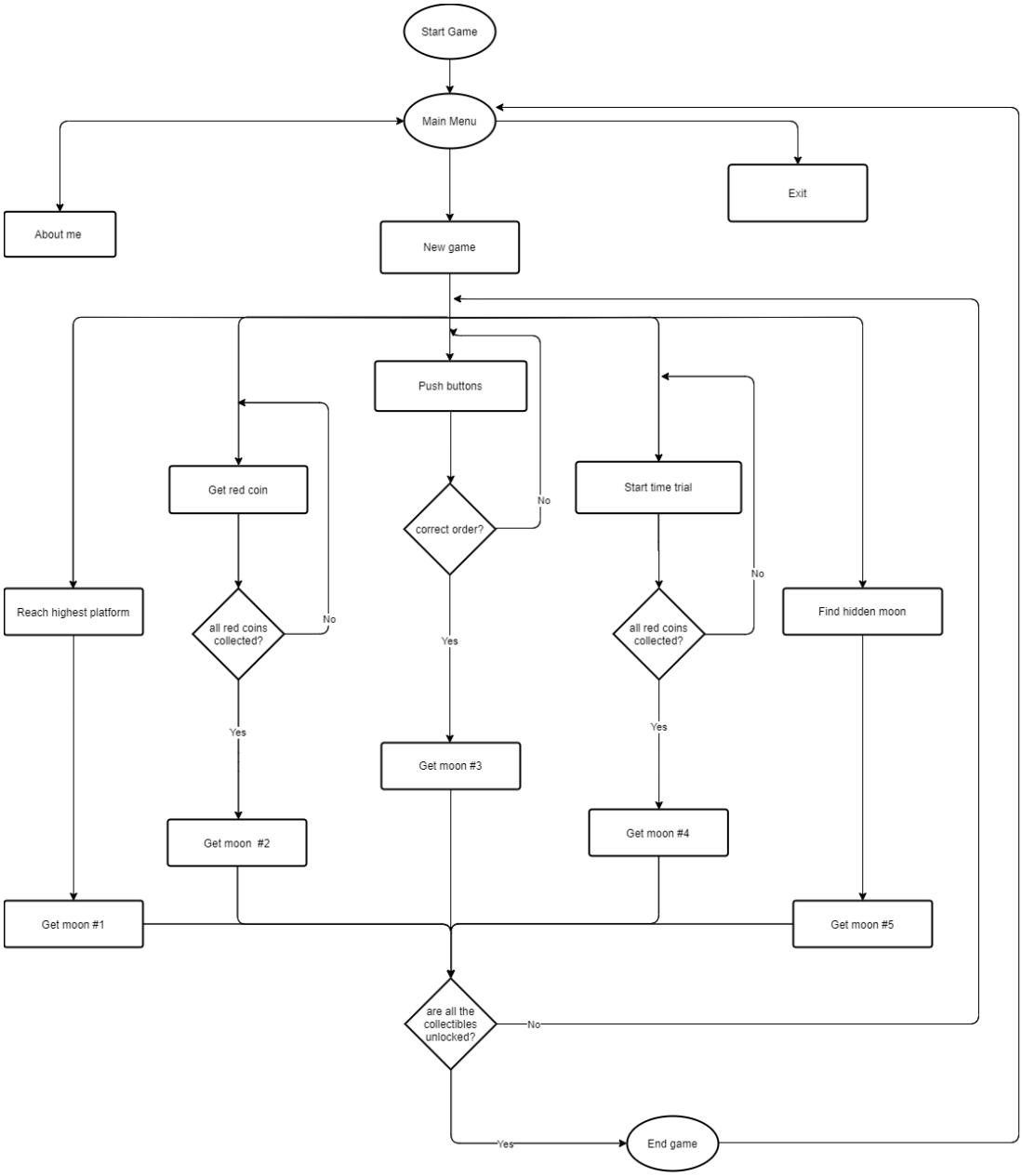

Figure 3.5: Flowchart

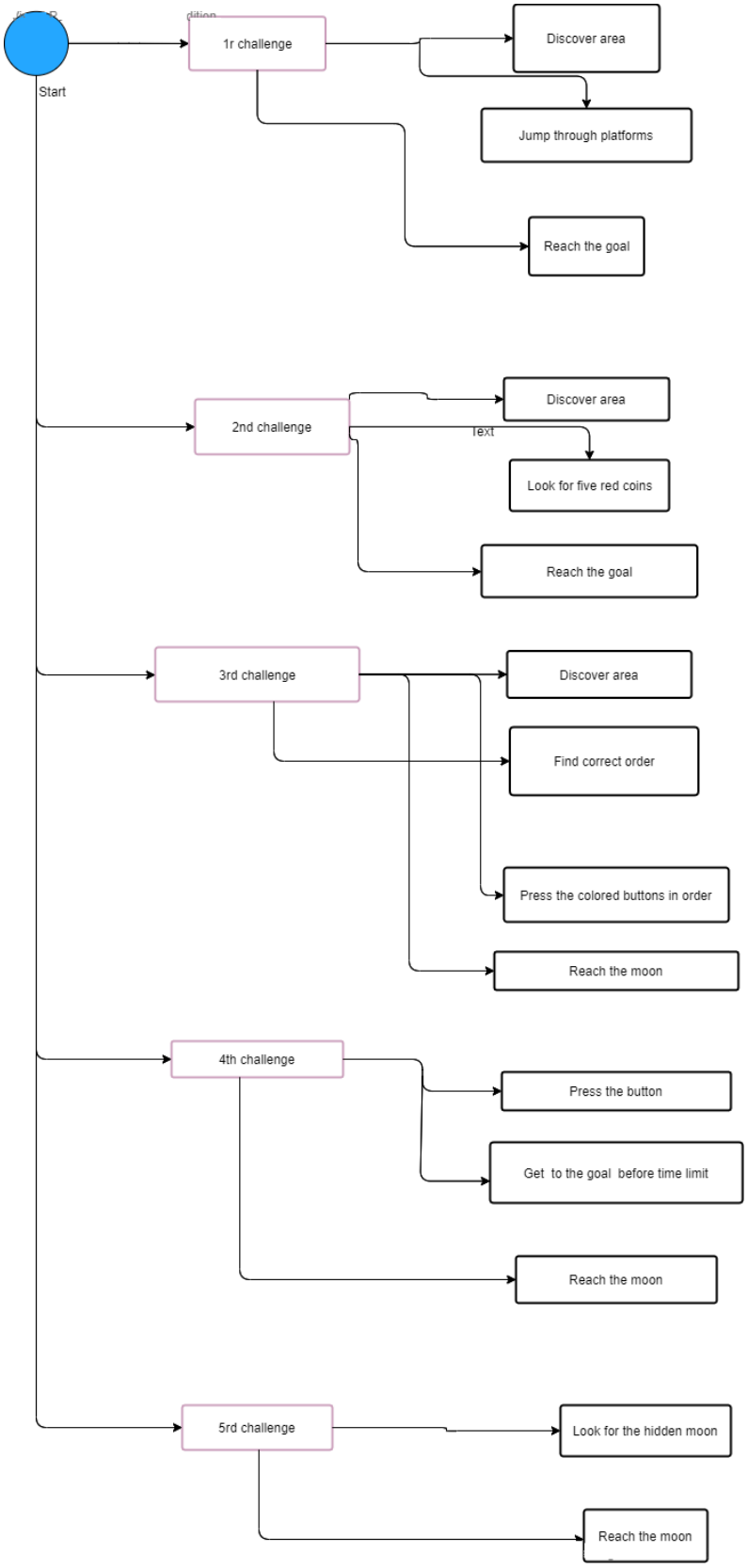

Figure 3.6: Flowgame

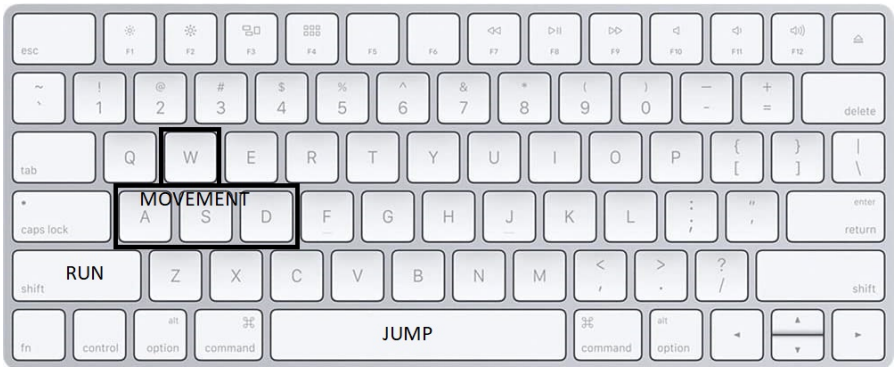

Figure 3.7: Keyboard controls

Figure 3.8: Gamepad controls

"drink me" potion.

When she wakes up a voice in her head tells her that she has to get the 5 hidden moons around her room so she can get back to normal size before morning comes. The moons are achieved by reaching high or hidden places as a reward.

Figure 3.9: TLOZ: Wind Waker

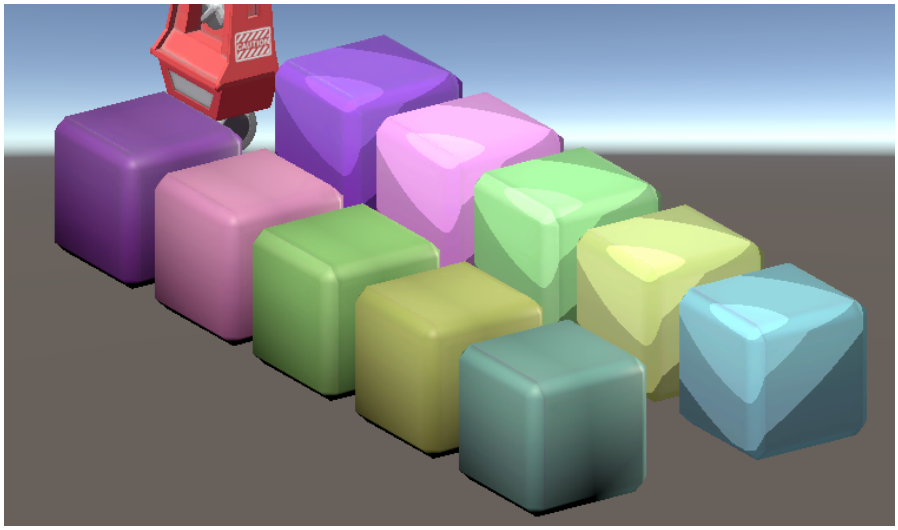

Figure 3.10: Unity: Universal Render Pipeline vs Toon Shader

Figure 3.11: Super Mario 64

Figure 3.12: Alice in Wonderland: Drink me potion

CHAPTER 4

Work Development and Results

This chapter shows the process of creation of the shader,the 3D models, the character and the building.

Contents

4.1 Work Development

4.1.1 CelShading

A cel-shader (or toon shader) is a type of non-realistic rendering designed to make 3D computer graphics appear to be flat by using less shading color instead of a shade gradient or tints and shades. The goal is to simulate typical comic book shading, this is achieved by restricting the diffuse color component of a fragment to only a certain number of possible values. The lighting values are calculated for each pixel and then assigned to a small number of discrete shadows to create the characteristic flat look, in which the lighter shadows and points appear to be blocks of color rather than smoothly blended.
The first 3D game to use this technique was Jet Set Radio [9], Jet Set Radio is an urban skating and street graffiti game created by Sega for the Dreamcast (see Figure 4.1). This technique consisted of, for the first time, hiding the vertices of the polygons and replacing them with black on their edges, and flat colors making the contrast between

the character and the background.

Figure 4.1: Jet Set Radio

To achieve this effect in Playroom and calcute the lighting, the shading model called Blinn-Phong will be used. Phong Model takes into consideration the viewer angle and how the light gets reflected from the surface. The Phong reflection describes how a surface reflects light as a combination of diffuse reflection from rough surfaces with specular reflection from shiny surfaces. It is based on Phong's informal observation that glossy surfaces have small, intense specular reflections, while opaque surfaces have large reflections that fall off more gradually.

Objects that are hit by light have a normal vector pointing 90 degrees from the surface of the object. The angle between the vector of the light hitting the object and the Normal of the object helps us to know at what angle the light source is hitting the object at a specific point. To achieve this, we use a dot product, although it must first be normalized, so the dot product will always be between [-1 and 1]. If the light source is hitting that point directly the product will be 1, if it is at 90 degrees it will be 0, and if it is backlit it will be -1.

4.1.2 How to create a toon shader in Unity

To do this in Unity, it's needed to create a new shader and a new material.
`Right click ->Shader -> Unlit shader` and name it ToonShader
`Right click -> New Material` create a new material to apply the toonshader to it.
Now its time to modify our new shader to make it look cartoon, so we open it and in struct appdata we create a normal `float3: NORMAL;` which will be the normal of our object. As well, we want to transform the normal from object space to world space, as the light's direction in Unity is provided in world space. We have to add the following line to the vertex shader: `o.worldNormal = UnityObjectToWorldNormal (v.normal);`

With the world normal now available in the fragment shader, we can compare it to the light's direction using the Dot Product. The dot product takes two vectors of any length and returns a number. When the vectors are parallel in the same direction and are vectors of length 1, this number is 1. When they are perpendicular, it returns 0. As you move a vector away from parallel toward perpendicular, the result of the scalar product it will go from 1 to 0 non-linearly. Note that when the angle between the vectors is greater than 90, the dot product will be negative.

The output of the vertex function is fed to the fragment function but its still need to add it to the v2f structure to ble able to use it, so in the struct v2f we create a Normal variable:

```
half3 worldNormal:  NORMAL;
```

Now I created a Cartoon function that receive as a parameter a float3 normal and a float3 that is the direction of the light:

```
float Cartoon(float3 normal, float 3 lightDirection)
float dotproduct = max(0.0, dot(normalize(normal), normalize(lightDirection));
```

and returns: `floor(dotproduct/0.3` and its divided by 0.3 to create color gradients (color groupings).

Now its needed to go to the fragment shader to add the shadows, so I created a variable Shad:

```
Shad* = Cartoon(i.worldNormal, WorldSpaceLightPos0.xyz);
```

```
PROPERTIES
```

Properties are values that can be changed to make materials look different. I created a float Brightness from 0 to 1 that will control hoe darker shadows are. Next I'm going to do the same with the other properties:

```
Strength("Strength".  Range(0,1)) = 0.5
```

and it makes the color look stronger or weaker depending the value. The next property is Color, that multiplies the cartoon function, changing the appearence:

```
Color("Color", COLOR) = (1,1,1,1)
```

and the last property I create is detail that changes the number of color groupings of shadows.

```
Detail("Detail", Range(0,1)) = 0.3
```

4.2 Props and 3D art

After researching 3D artists workflow for game developing, the game assets were created by the following process. First, it is required any 3D modeling program such as Blender, then its common to create a lowpoly version, which will be the one that will be used in the game engine. Next, a high-poly version is created, with more polygons and sometimes with more detail, the textures of the highpoly version are then baked in the lowpoly

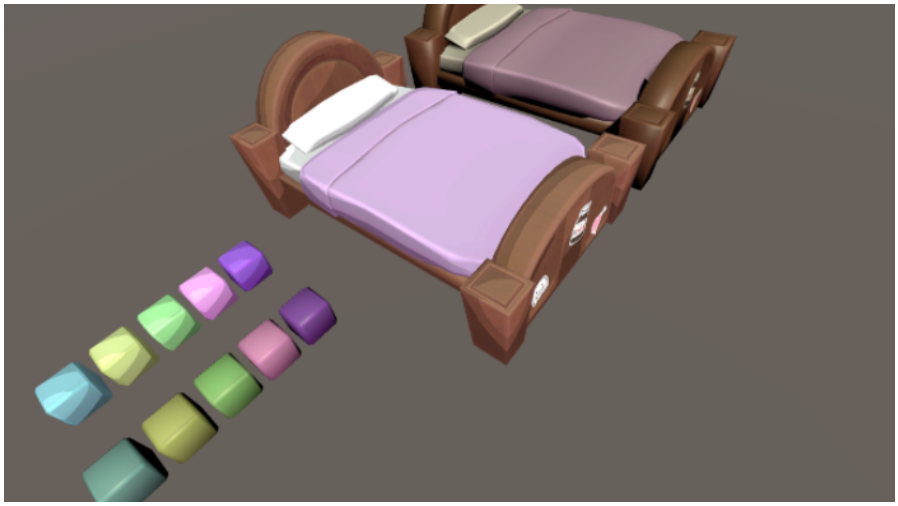

Figure 4.2: Cel-Shading Comparation

model, thus simulating the detail in the textures.(see Figure 4.7) The left model is the lowpoly version, which it's the one that will be placed in the game engine. The right model is the highpoly version, which was sculpted in [7], with a handmade wood look. Then we bake the highpoly textures into the lowpoly to make the models look alike. All the models in the game are made by me (see Figure 4.6),(see Figure 4.3), (see Figure 4.4) and (see Figure 4.5) are some examples.

The left model is the lowpoly version, which it's the one that will be placed in the game engine. The right model is the highpoly version, which was sculpted in [7], with a handmade wood look.

Then the highpoly textures are baked into the lowpoly to make the models look alike. This procedure is done using Substance Painter for texture baking, in this way, the resulting models have fewer polygons and simulates details. (see Figure 4.8)

4.2.1 Texturing

Substance Painter is a 3D texture editor by Adobe. All the assets are textured in substance painter by using a custom smart material. A smart material is preset for a whole layerstack, it could be seen as a folder with a few layers, that has been saved and can be re-used elsewhere. A correctly-made Smart Material contains only effects and layers that work on every mesh (with baked maps), such as Generators and Filters. Generators are substances that generate a mask or a material based on the mesh topology using the additional maps setup in the TextureSet Settings, each generator has a set of parameters allowing you to fine-tune the resulting mask. Filter Effects are substances that transforms the content of a layer or mask.

Figure 4.3: 3D models

Figure 4.4: 3D models

Figure 4.5: 3D models

Figure 4.6: 3D models

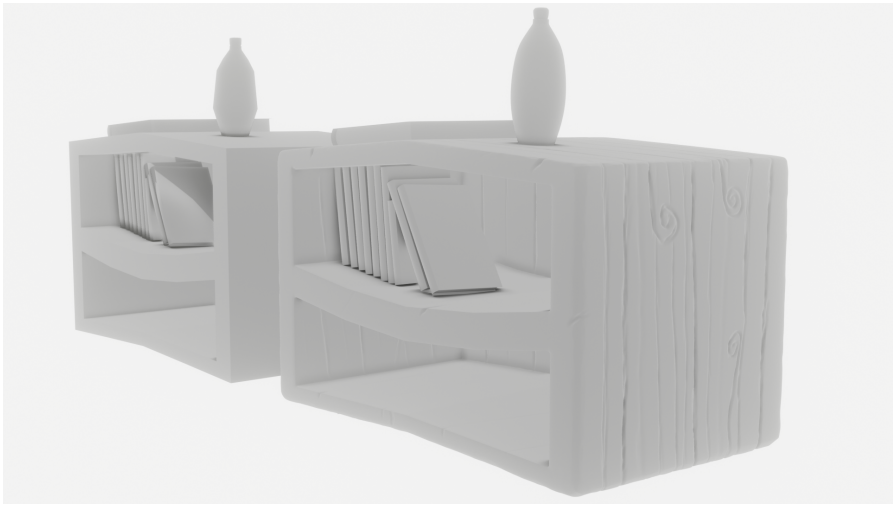

Figure 4.7: Lowpoly and Highpoly

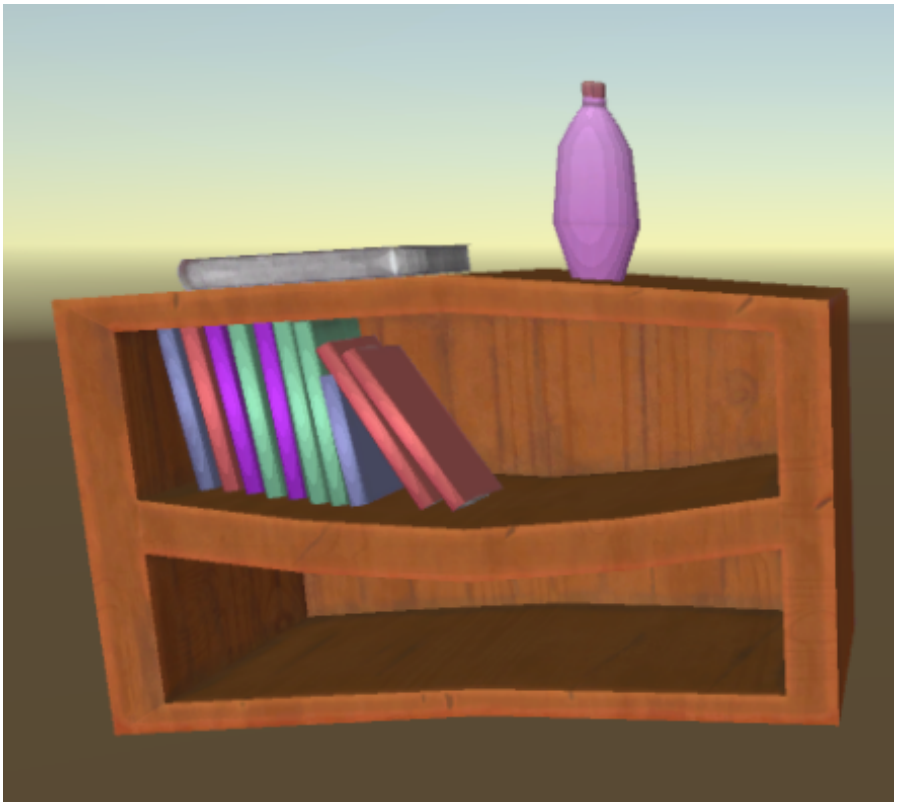

Figure 4.8: Baked

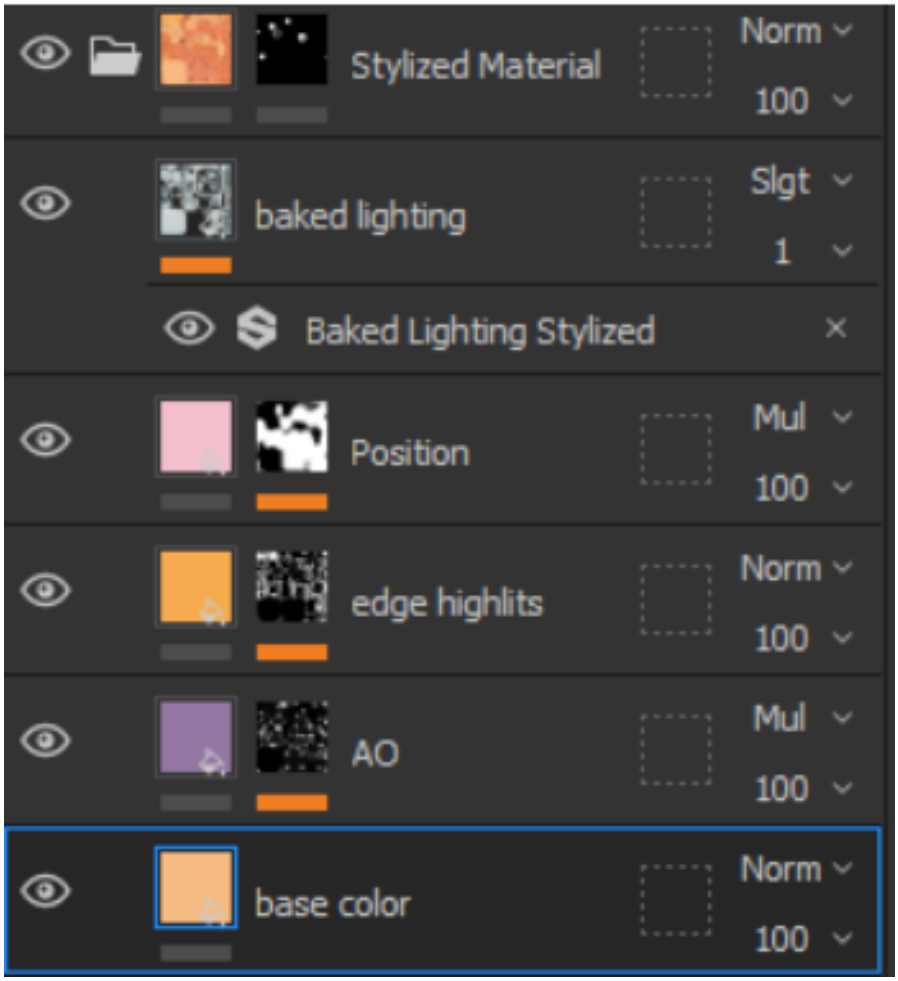

Figure 4.9: Smart Material

Smart Material

A stylized material is created by adding generators and filters to it (see Figure 4.9) First of all, the first layer is base color. Just the color of the object. The AO layer is an ambient occlusion layer. The ambient occlusion channel allows painting details in the ambient shadows of an object. It can be used to add AO details coming from Materials, or simply fix manually baking errors when needed. The Edge highlights layer use a curvature generator. It uses the curvature of the mesh to paint the edges of the object. Works very well with a cartoon style because it adds color variation. The layer position uses the position generator, which uses the position map to make a color gradient, typical of cartoon drawing (see Figure 4.10)

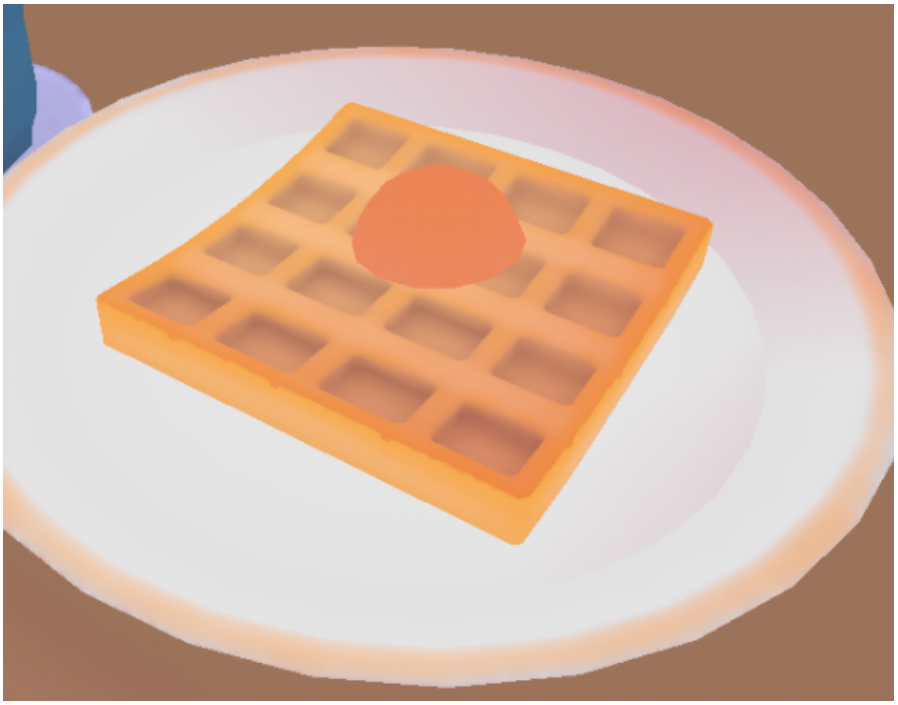

Figure 4.10: Waffle

4.2.2 Main Character

The main character was modeled in Blender and sculpted in Zbrush, I wanted it to look like the style of The Legend Of Zelda: Wind waker, so the face was not sculpted, but rather added for texture.

The character model is simple but effective, the animations were downloaded and applied from [1]. It was my first time making a 3D character, so I had no idea how to do it. In spite of everything I think she has turned out very well and I am very happy with how she is in the game.

4.3 Results

The results were roughly as expected. In my opinion, even knowing that a game has a lot of work behind it, I have underestimated it: modeling, shader programming, and gameplay took more effort than I had planned to carry out. Still, I am happy with the result and I learned a lot.

Chapter 5

Conclusions and Future Work

Contents

5.1 Conclusions

I really enjoyed the process of creating the Shader, as I did a lot of research and learned a lot on my own. In addition, the fact of creating a video game from scratch by myself has made me realize all the work that goes into it since it has been the first time that I have created all the 3D assets of a video game myself. The biggest challenge of this job was learning how shaders work in Unity and learning about 3D modeling. I think I have learned from both, and although the models are not the best possible, I have learned a lot about UV's, textures, substance painter and their materials, and how the video game industry workflow works. It is the first time that I use latex, so learning how to use it has seemed interesting and very useful.

5.2 Future work

I think that with this work I have managed to learn everything I wanted: how to make 3D assets for video games, how to make a shader, the importance of this within a graphics engine and the workload that is to make a video game from scratch. My goal for the future is that this game can serve as a portfolio for me, and from there continue to improve in everything to be able to dedicate professionally to it one day.

Bibliography

[1] Adobe. Mixamo. https://www.mixamo.com. Accessed: 2021-06-30.

[2] Adobe. Photoshop. https://www.adobe.com/es/products/photoshop.html. Accessed: 2021-06-28.

[3] Adobe. Substance painter. https://www.substance3d.com/products/substance-painter/. Accessed: 2021-05-28.

[4] Lewis Caroll. *Alice in Wonderland*. Macmillan Publishers, 1865.

[5] Nintendo. The legend of zelda: Wind waker. https://www.nintendo.es/Juegos/Wii-U/The-Legend-of-Zelda-The-Wind-Waker-HD-765386.html. Accessed: 2021-05-24.

[6] Nintendo. Super mario 64. https://www.nintendo.es/Juegos/Nintendo-64/Super-Mario-64-269745.html. Accessed: 2021-05-24.

[7] Pixologic. Zbrush. https://pixologic.com/features/. Accessed: 2021-06-16.

[8] Ton Roosendaal. Blender. https://www.blender.org. Accessed: 2021-05-28.

[9] Smilebit. Jetsetradio. https://www.sega.es/games/jet-set-radio. Accessed: 2021-06-28.

[10] Unity Technologies. Unity. https://unity.com/es. Accessed: 2021-06-28.

[11] Wikipedia. Functional requirements. http://en.wikipedia.org/wiki/Functional_requirements. Accessed: 2019-02-28.

.1 Bibliography

[1] Roystan toon shader https://roystan.net/articles/toon-shader.html

[2] 3dEx youtube channel https://www.youtube.com/c/irvin390/videos

[3] Unity tutorials https://learn.unity.com/

[4] 80lvl tutorials https://80.lv/articles/environment-art/

[5] Stylized Station youtube channel https://www.youtube.com/channel/UC7cmH–tFhYduIshTKzQUJQ